CSE 332: Data Structures
Priority Queues – Binary Heaps
Richard Anderson
Spring 2016

Recall Queues
- FIFO: First-In, First-Out
  - Print jobs
  - File serving
  - Phone calls and operators
  - Lines at the Department of Licensing…

Priority Queues
Prioritize who goes first – a priority queue:
- treat ER patients in order of severity
- route network packets in order of urgency
- operating system can favor jobs of shorter duration or those tagged as having higher importance
- Greedy optimization: “best first” problem solving

Priority Queue ADT
- Need a new ADT
- Operations: Insert an Item, Remove the “Best” Item

Potential implementations

<table>
<thead>
<tr>
<th></th>
<th>insert</th>
<th>deleteMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Array)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted list (Linked-List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Search Tree (BST)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Binary Heap data structure

• **binary heap** (a kind of binary tree) for priority queues:
  – $O(\log n)$ worst case for both insert and deleteMin
  – $O(1)$ average insert

• It’s optimized for priority queues. Lousy for other types of operations (e.g., searching, sorting)

More Tree Terminology

- **depth(B):**
- **height(G):**
- **height(T):**
- **degree(B):**
- **branching factor(T):**
- **n-ary tree:**

Binary Heap Properties

A binary heap is a binary tree with two important properties that make it a good choice for priority queues:

1. **Completeness**
2. **Heap Order**

**Note:** we will sometimes refer to a binary heap as simply a “heap”.

Completeness Property

• A binary heap is a **complete** binary tree:
  – a binary tree with all levels full, except possibly the bottom level, which is filled left to right

Examples:

Heights of a **complete** binary tree with $n$ nodes?

Heap **Order** Property

**Heap order property:** For every non-root node $X$, the value in the parent of $X$ is less than (or equal to) the value in $X$.
Heap Operations

- Main ops: insert, deleteMin
- Key is to maintain
  - Completeness
  - Heap Order
- Basic idea is to propagate changes up/down the tree, fixing order as we go

Heap – insert(val)

Basic Idea:
1. Put val at last leaf position
2. Percolate up by repeatedly exchanging node with parent as long as needed

Insert: percolate up

Heap – deleteMin

Basic Idea:
1. Remove min element
2. Put “last” leaf node value at root
3. Find smallest child of node
4. Swap node with its smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

DeleteMin: percolate down

DeleteMin: percolate down
Representing Complete Binary Trees in an Array

From node $i$:
- left child:
- right child:
- parent:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Why use an array?

DeleteMin Code

```java
public Object deleteMin() {
    assert(!isEmpty());
    returnVal = Heap[1];
    size--;
    newPos = percolateDown(1, Heap[size + 1]);
    Heap[newPos] = Heap[size + 1];
    return returnVal;
}
```

Insert Code

```java
public void insert(Object o) {
    assert(!isFull());
    size++;
    newPos = percolateUp(size, o);
    Heap[newPos] = o;
}
```

runtime:

(Java code in book)

More Priority Queue Operations

decreaseKey(nodePtr, amount):
- given a pointer to a node in the queue, reduce its priority

Binary heap: change priority of node and ________________

increaseKey(nodePtr, amount):
- given a pointer to a node in the queue, increase its priority

Binary heap: change priority of node and ________________

Why do we need a pointer? Why not simply data value?

Worst case running times?
More Priority Queue Operations

**remove(objPtr):**
given a pointer to an object in the queue, remove it

Binary heap: ______________________________________

**findMax():**
Find the object with the highest value in the queue

Binary heap: ______________________________________

Worst case running times?

More Binary Heap Operations

**expandHeap():**
If heap has used up array, copy to new, larger array.
• Running time:

**buildHeap(objList):**
Given list of objects with priorities, fill the heap.
• Running time:

We do better with `buildHeap`...

Building a Heap: Take 1

BuildHeap: Floyd’s Method

Add elements arbitrarily to form a complete tree. Pretend it’s a heap and fix the heap-order property!

Red nodes need to percolate down

Key idea: fix red nodes from bottom-up

Finally…
Buildheap pseudocode

```java
private void buildHeap() {
    for (int i = currentSize/2; i > 0; i--)
        percolateDown(i);
}
```

Buildheap Analysis

- \(\frac{n}{4}\) nodes percolate at most 1 level
- \(\frac{n}{8}\) percolate at most 2 levels
- \(\frac{n}{16}\) percolate at most 3 levels...

```
```